Project Milestone

4/12/2026
Parallel LZW and LZ77 Compression

Summarization of Work Done So Far

Our project goal is to compare the LZW and LZ77 compression algorithms with respect to compression ratio, runtime, and speedup at higher core counts.

LZ77. We have created a working parallel implementation that chunks the data and assigns work to threads via the OpenMP library. We are currently experimenting with different window sizes across file types and thread counts, as well as alternate parallelization strategies. On a single core the compression ratio is approximately 0.4× the original file size; however, this ratio degrades as the thread count increases. Speedup is very linear on the GHC machines with fast runtimes.

LZW. We have achieved performance comparable to Unix's `compress` utility (single-threaded) with our baseline serial LZW implementation, both in compression speed and compression ratio. For the parallel version we also use OpenMP to process different file sections in parallel. On very large files (e.g. 1 GB Wikipedia data) our parallel code achieves roughly 5× speedup with 8 GHC cores, though this parallelism suffers on small files where overheads dominate.

Goals and Deliverables

We have made good progress toward our initial targets, even exceeding them in some areas. We have working serial and parallel implementations of both LZ77 and LZW, and are confident we can deliver the goals from our project proposal.

Summary of goals:

- ∼6× speedup on 8 cores (GHC) for both LZ77 and LZW parallel implementations.
  - The serial LZW speed should be competitive with the Unix `compress` algorithm available on GHC machines and macOS.
  - We are targeting Zstandard (Zstd) as a reference for our LZ77-based approach. Since Zstd is a widely adopted standard that includes steps beyond LZ77 in its pipeline, our goals are less ambitious and do not aim to match it in speed or compression ratio.
- Conservatively, ∼32× speedup on 64 cores (PSC) for both LZW and LZ77.

LZ77 Goal

Parallelize LZ77 sufficiently to compare and compete with other LZ compression algorithms while maintaining the simplicity of the sliding-window approach.

LZW Goal

Explore more advanced parallelism strategies beyond the current naive blocked approach to achieve better speedup. Additionally, improve performance on smaller files where overhead currently dominates, and address compression-ratio degradation on inputs with highly repeated patterns (e.g. novels).

Schedule

Dates	Task(s)
4/13 – 4/16	*Alex:* Explore using a parallel dictionary for LZW beyond the naive blocked approach. *Jack:* Experiment with CUDA threads for string matching in LZ77.
4/17 – 4/19	*Alex:* Test LZW on small files and high core counts; improve scaling. *Jack:* Focus on better compression ratios after achieving improved runtime with CUDA-threaded LZ77.
4/20 – 4/23	*Both:* Tune both algorithms. Compare LZ77 and LZW against each other and explore pushing each toward its extreme.
4/24 – 4/26	*Jack:* Finalize LZ77 code. *Alex:* Finalize LZW code. *Both:* Ensure the codebase is well organized and documented.
4/27 – 4/30	*Both:* Finalize report, complete final testing, and generate graphs. *Both:* Update website with findings. *Both:* Create and rehearse the presentation poster.

Preliminary Results

LZ77 Preliminary Results

The parallel LZ77 implementation on the Canterbury corpus scales well in terms of speedup (Figure 1), with only a slight decrease in compression ratio across thread counts (Figure 2). However, the raw runtime is still poor compared to other LZ compression algorithms; reducing this gap is a key focus for the next two weeks.

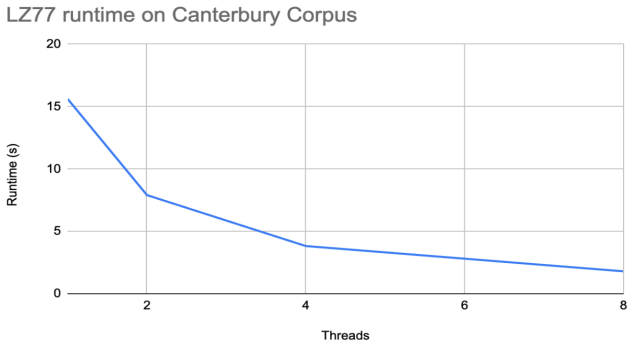

Figure 1: LZ77 runtime on the Canterbury corpus as a function of thread count.

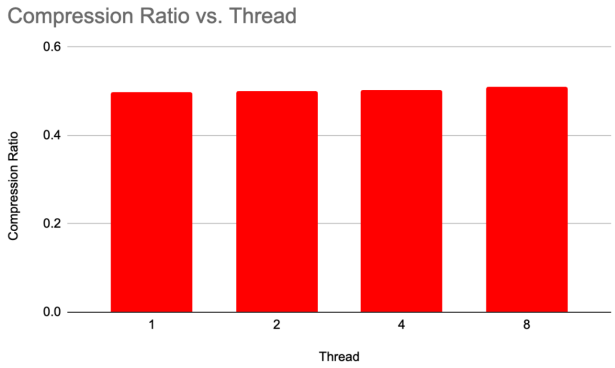

Figure 2: LZ77 compression ratio on the Canterbury corpus across thread counts.

LZW Preliminary Results

As shown in Figure 3, the naive chunked approach scales quite well up to 8 cores on the large Enwik9 dataset. Some performance deterioration is observed moving from 4 to 8 cores, and we expect this trend to continue at 16, 32, and 64 cores. The Unix `compress` compression ratio was 0.449, while our LZW achieved 0.430.

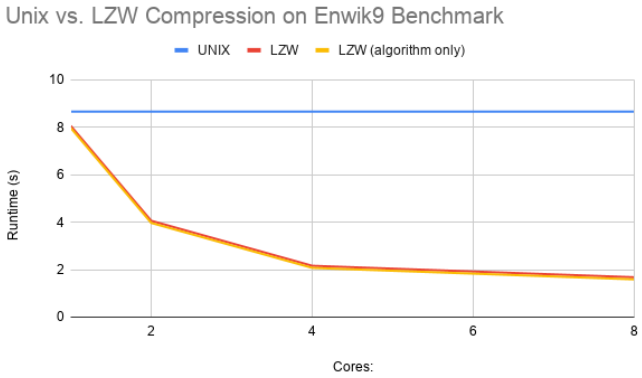

Figure 3: Unix `compress` vs. parallel LZW runtime on the Enwik9 benchmark.

Poster Session Plan

We plan to present comparison graphs of runtime and compression ratio for both serial and parallel implementations at $n_{procs} \in \{1, 2, 4, 8\}$, with higher core counts possible if PSC time is available. For LZW we will also include reference numbers from Unix `compress`. Results will be presented graphically.

Issues and Concerns

We have not encountered any specific problems meeting deadlines or making progress on either algorithm.

A concern is whether our final results will be sufficiently compelling. Our goal is to explore different parallelization strategies for two related algorithms and highlight meaningful differences between them. We hope to achieve both good compression ratios and strong speedup so that the comparison is interesting.

Specific risks include:

- Poor scaling on small files where parallelism overhead dominates.
- Difficulty achieving sufficient work per core at high thread counts (64+ cores on PSC).
- For LZW in particular, fast concurrent access to a shared dictionary structure is the most technically challenging aspect and the most likely

bottleneck for hitting our scaling targets.